EXHIBIT 8

ConnectionManager:1
Service Template Version 1.01

For Universal Plug and Play Version 1.0
Status: Standardized DCP
Date: June 25, 2002

This Standardized DCP has been adopted as a Standardized DCP by the Steering Committee of the UPnP Forum, pursuant to Section 2.1(c)(ii) of the UPnP Membership Agreement. UPnP Forum Members have rights and licenses defined by Section 3 of the UPnP Membership Agreement to use and reproduce the Standardized DCP in UPnP Compliant Devices. All such use is subject to all of the provisions of the UPnP Membership Agreement.

Authors	Company
Shannon Chan	Microsoft
Alec Dara-Abrams	Sony Electronics
Mike Dawson	OpenGlobe
John Kai Fu	Pioneer
Fernando Matsubara	Mitsubishi Electric
Jean Moonen	Philips Electronics
Yasser Rasheed	Intel
Dale Sather	Microsoft
Eugene Shteyn	Philips Electronics

Contents

List of Tables

1. Overview and Scope

This service definition is compliant with the UPnP Device Architecture version *1.0*.

This service-type enables modeling of streaming capabilities of A/V devices, and binding of those capabilities between devices. Each device that is able to send or receive a stream according to the UPnP AV device model [ref to dev model] will have 1 instance of the ConnectionManager service. This service provides a mechanism for control points to:

1. Perform capability matching between source/server devices and sink/renderer devices,
2. Find information about currently ongoing transfers in the network,
3. Setup and teardown connections between devices (when required by the streaming protocol).

The ConnectionManager service is generic enough to properly abstract different kinds of streaming mechanisms, such as HTTP-based streaming, RTSP/RTP-based and 1394-based streaming.

The ConnectionManager enables control points to abstract from physical media interconnect technology when making connections. The term 'stream' used in this service template refers to both analog and digital data transfer.

1.1. External dependencies

This standard references the following external documents:

- Hypertext Connection Protocol – HTTP/1.1 (http://www.ietf.org/rfc/rfc2616.txt)
- MIME (Multipurpose Internet Mail Extensions) (http://www.ietf.org/rfc/rfc1341.txt)
- Real Time Streaming Protocol (RTSP) (http://www.ietf.org/rfc/rfc2326.txt)
- Realtime Transport Protocol (RTP) (http://www.ietf.org/rfc/rfc1889.txt)
- IEC 61883 Consumer Audio/Video Equipment – Digital Interface - Part 1 to 5 (http://www.iec.ch/).
- IEC-PAS 61883 Consumer Audio/Video Equipment – Digital Interface - Part 6 (http://www.iec.ch/).

2. Service Modeling Definitions

2.1. ServiceType

The following service type identifies a service that is compliant with this template:

urn:schemas-upnp-org:service:ConnectionManager:1

2.2. State Variables

Table 1: State Variables

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value	Default Value	Eng. Units
SourceProtocolInfo	R	string	CSV[2] (string)		
SinkProtocolInfo	R	string	CSV (string)		
CurrentConnectionIDs	R	string	CSV (ui4)		
A_ARG_TYPE_ConnectionStatus	R	string	“OK”, “ContentFormatMismatch”, “InsufficientBandwidth”, “UnreliableChannel”, “Unknown”	n/a	n/a
A_ARG_TYPE_ConnectionManager	R	string		n/a	n/a
A_ARG_TYPE_Direction	R	string	“Output”, “Input”	n/a	n/a
A_ARG_TYPE_ProtocolInfo	R	string		n/a	n/a
A_ARG_TYPE_ConnectionID	R	i4		n/a	n/a
A_ARG_TYPE_AVTransportID	R	i4		n/a	n/a
A_ARG_TYPE_RcsID	R	i4		n/a	n/a

[1] R = Required, O = Optional, X = Non-standard.

2.2.1. SourceProtocolInfo

This variable contains a comma-separated list of information on protocols this ConnectionManager supports for “sourcing” (sending) data, in its current state. Besides the traditional notion of the term ‘protocol’, the protocol-related information provided by the connection also contains other information such as supported content formats. See the Theory of Operation (Section 2.5.2) for a general discussion on the notion of protocol info. See the table in Section 2.5.2 for specific allowed values for this state variable.

[2] CSV stands for Comma-Separated Value list. The type between brackets denotes the UPnP data type used for the elements inside the list. CSV is defined more formally in the ContentDirectory service template.

2.2.2. SinkProtocolInfo

This variable contains a comma-separated list of information on protocols this ConnectionManager supports for "sinking" (receiving) data, in its current state. The format and allowed value list are the same as for the SourceProtocolInfo state variable.

2.2.3. CurrentConnectionIDs

Comma-separated list of references to current active Connections. This list may change without explicit actions invoked by Control points, for example, by out-of-band cleanup or termination of finished connections.

If optional action PrepareForConnection is not implemented then this state variable should be set to "0".

2.2.4. A_ARG_TYPE_ConnectionStatus

The current status of the Connection referred to by variable A_ARG_TYPE_ConnectionID. This status may change dynamically due to changes in the network.

2.2.5. A_ARG_TYPE_ConnectionManager

This state variable is introduced to provide type information for the "PeerConnectionManager" parameter in actions PrepareForConnection and GetCurrentConnectionInfo. A ConnectionManager reference takes the form of a UDN/Service-Id pair (the slash is the delimiter). A control point can use UPnP discovery (SSDP) to obtain a ConnectionManager's description document from the UDN. Subsequently, the ConnectionManager's service description can be obtained by using the serviceId part of the reference.

2.2.6. A_ARG_TYPE_Direction

This state variable is introduced to provide type information for the "Direction" parameter in action PrepareForConnection.

2.2.7. A_ARG_TYPE_ProtocolInfo

This state variable is introduced to provide type information for the "Protocol" parameter in actions PrepareForConnection and GetCurrentConnectionInfo.

2.2.8. A_ARG_TYPE_ConnectionID

This state variable is introduced to provide type information for the "ConnectionID" parameter in actions: PrepareForConnection, ConnectionComplete and GetCurrentConnectionInfo.

2.2.9. A_ARG_TYPE_AVTransportID

This state variable is introduced to provide type information for the "AVTransportID" parameter in actions: PrepareForConnection and GetCurrentConnectionInfo. It identifies a logical instance of the AVTransport service associated with a Connection. See [ref to Device Model] for more information.

2.2.10.A_ARG_TYPE_RcsID

This state variable is introduced to provide type information for the "RcsID" parameter in actions: PrepareForConnection and GetCurrentConnectionInfo. It identifies a logical instance of the Rendering Control service associated with a Connection. See [ref to Device Model] for more information.

2.3. Eventing and Moderation

Table 2: Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
SourceProtocolInfo	Yes	No	n/a	n/a	n/a
SinkProtocolInfo	Yes	No	n/a	n/a	n/a
CurrentConnectionIDs	Yes	No	n/a	n/a	n/a

[1] Determined by N, where Rate = (Event)/(N secs).
[2] (N) * (allowedValueRange Step).

2.4. Actions

Immediately following this table is detailed information about these actions, including short descriptions of the actions, the effects of the actions on state variables, and error codes defined by the actions.

Table 3: Actions

Name	Req. or Opt. [1]
GetProtocolInfo	R
PrepareForConnection	O
ConnectionComplete	O
GetCurrentConnectionIDs	R
GetCurrentConnectionInfo	R

[1] R = Required, O = Optional, X = Non-standard.

2.4.1. GetProtocolInfo

Returns the protocol-related info that this ConnectionManager supports in its current state, as a comma-separate list of strings according to Table 2.

2.4.1.1. Arguments

Table 4: Arguments for GetProtocolInfo

Argument	Direction	relatedStateVariable
Source	OUT	SourceProtocolInfo
Sink	OUT	SinkProtocolInfo

2.4.1.2. Dependency on State (if any)

2.4.1.3. Effect on State (if any)

2.4.1.4. Errors

None.

2.4.2. PrepareForConnection

This action is used to allow the device to prepare itself to connect to the network for the purposes of sending or receiving media content (e.g. a video stream). The RemoteProtocolInfo parameter identifies the protocol, network, and format that should be used to transfer the content. Its value corresponds to one of the ProtocolInfo entries returned by the GetProtocolInfo() action from the remote device. If the remote device does not implement GetProtocolInfo(), then the RemoteProtocolInfo parameter should be set to one of the ProtocolInfo entries returned by the GetProtocolInfo() action on the local device.

2.4.2.1. Arguments

Table 5: Arguments for PrepareForConnection

Argument	Direction	relatedStateVariable
RemoteProtocolInfo	IN	A_ARG_TYPE_ProtocolInfo
PeerConnectionManager	IN	A_ARG_TYPE_ConnectionManager
PeerConnectionID	IN	A_ARG_TYPE_ConnectionID
Direction	IN	A_ARG_TYPE_Direction
ConnectionID	OUT	A_ARG_TYPE_ConnectionID
AVTransportID	OUT	A_ARG_TYPE_AVTransportID
RcsID	OUT	A_ARG_TYPE_RcsID

2.4.2.2. Dependency on State (if any)

2.4.2.3. Effect on State (if any)

Prepares the device to stream content to or from the specified peer ConnectionManager, according to the specified direction and protocol information. The PeerConnectionManager identifies the ConnectionManager service on the other side of the connection. The PeerConnectionID identifies the specific connection on that ConnectionManager service. This information allows a control point to "link" a connection on device A to the corresponding connection on device B, via action GetCurrentConnectionInfo. If the PeerConnectionID is not known by a control point (e.g., this is the first of the two PrepareForConnection actions, or the peer device doesn't implement PrepareForConnection) then this value should be set to reserved value '-1'.

Returns a locally unique ID for the established Connection (ConnectionID parameter), and adds that ID to state variable CurrentConnectionIDs. This ID might be used by a control point to manually terminate the established Connection through (optional) action ConnectionComplete. It can also be used to retrieve information associated with the Connection via action GetCurrentConnectionInfo. Value -1 is reserved, and should not be returned.

Optionally returns a virtual instance ID of a local AVTransport service (AVTransportID parameter). This ID should be passed as an input parameter to the local AVTransport service action invocations. If the returned ID is –1 (reserved value), then there is no AVTransport service on this device that can be used to control the established connection. This is dependent on the 'push' or 'pull' nature of the streaming protocol.

Optionally returns a virtual instance ID of a local RenderingControl service (RcsID parameter). This ID should be passed as an input parameter to the local RenderingControl service action invocations. If the returned ID is –1 (reserved value), then there is no RenderingControl service on this device, for example, because the device is a source device (MediaServer) rather than a sink device (MediaRenderer).

Due to local restrictions on the device running the ConnectionManager, variable "ProtocolInfo' may change (e.g., certain physical ports on the device are not available anymore for new connections) as a result of this action.

2.4.2.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	One of following: not enough IN arguments, too many IN arguments, no IN argument by that name, one or more IN arguments are of the wrong data type. See also the Universal Plug and Play Device Architecture.
707	Not in network	The connection cannot be established because the ConnectionManagers are not part of the same physical network.
701	Incompatible protocol info	The connection cannot be established because the protocol info parameter is incompatible.
702	Incompatible directions	The connection cannot be established because the directions of the involved ConnectionManagers (source/sink) are incompatible.
703	Insufficient network resources	The connection cannot be established because there are insufficient network resources (bandwidth, channels, etc.).
704	Local restrictions	The connection cannot be established because of local restrictions in the device. This might happen, for example, when physical resources on the device are already in use by other connections.
705	Access denied	The connection cannot be established because the client is not permitted to access the specified ConnectionManager.

2.4.3. ConnectionComplete

A control point should call the ConnectionComplete action for all connections that it created via PrepareForConnection to ensure that all resources associated with the connection are freed up. In addition, a ConnectionManager may implemented 'automatic' or 'autonomous' closing of connections, in a protocol and vendor-specfic way, see Appendix A for details.

2.4.3.1. Arguments

Table 6: Arguments for ConnectionComplete

Argument	Direction	relatedStateVariable
ConnectionID	IN	A_ARG_TYPE_ConnectionID

2.4.3.2. Dependency on State (if any)

2.4.3.3. Effect on State (if any)

Remove the connection referenced by parameter ConnectionID by modifying state variable CurrentConnectionIDs, and (if necessary) perform any protocol-specific cleanup actions such as releasing network resources. See the Appendix for protocol specifics.

Due to local restrictions on the device running the ConnectionManager, variables "SourceProtocolInfo' and "SinkProtocolInfo' may change (e.g., certain physical ports on the device are freed up for new connections).

2.4.3.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	One of following: not enough IN arguments, too many IN arguments, no IN argument by that name, one or more IN arguments are of the wrong data type. See also the Universal Plug and Play Device Architecture.
706	Invalid connection reference	The connection reference argument does not refer to a valid connection established by this service.

2.4.4. GetCurrentConnectionIDs

Returns a comma-separated list of ConnectionIDs of currently ongoing Connections. A ConnectionID can be used to manually terminate a Connection via action ConnectionComplete, or to retrieve additional information about the ongoing Connection via action GetCurrentConnectionInfo.

2.4.4.1. Arguments

Table 7: Arguments for GetCurrentConnectionIDs

Argument	Direction	relatedStateVariable
ConnectionIDs	OUT	CurrentConnectionIDs

2.4.4.2. Dependency on State (if any)

2.4.4.3. Effect on State (if any)

2.4.4.4. Errors

None.

2.4.5. GetCurrentConnectionInfo

Returns associated information of the connection referred to by the 'ConnectionID' parameter. The 'AVTransportID' and 'PeerConnectionManager' parameters may be NULL (empty string) in cases where the connection has been setup completely out of band, e.g., not involving a PrepareForConnection action.

If optional action PrepareForConnection is not implemented then (limited) connection information can be retrieved for ConnectionID 0. The device should return all known information:

- RcsID should be 0 or –1
- AVTransportID should be 0 or –1
- ProtocolInfo should contain accurate information if it is known, other it should be NULL (empty string)
- PeerConnectionManager should be NULL (empty string)
- PeerConnectionID should be –1
- Direction should be Input or Output
- Status should be OK or Unknown

2.4.5.1. Arguments

Table 8: Arguments for GetCurrentConnectionInfo

Argument	Direction	relatedStateVariable
ConnectionID	IN	A_ARG_TYPE_ ConnectionID
RcsID	OUT	A_ARG_TYPE_RcsID
AVTransportID	OUT	A_ARG_TYPE_ AVTransportID
ProtocolInfo	OUT	A_ARG_TYPE_ ProtocolInfo
PeerConnectionManager	OUT	A_ARG_TYPE_ConnectionManager
PeerConnectionID	OUT	A_ARG_TYPE_ConnectionID
Direction	OUT	A_ARG_TYPE_Direction
Status	OUT	A_ARG_TYPE_ConnectionStatus

2.4.5.2. Dependency on State (if any)

2.4.5.3. Effect on State (if any)

2.4.5.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	One of following: not enough IN arguments, too many IN arguments, no IN argument by that name, one or more IN arguments are of the wrong data type. See also the Universal Plug and Play Device Architecture.
706	Invalid connection reference	The connection reference argument does not refer to a valid connection established by this service.

2.4.6. Common Error Codes

The following table lists error codes common to actions for this service type. If an action results in multiple errors, the most specific error should be returned.

Table 9: Common Error Codes

errorCode	errorDescription	Description
401	Invalid Action	See UPnP Device Architecture section on Control.
402	Invalid Args	See UPnP Device Architecture section on Control.
404	Invalid Var	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
600-699	TBD	Common action errors. Defined by UPnP Forum Technical Committee.
701-799		Common action errors defined by the UPnP Forum working committees.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.5. Theory of Operation

2.5.1. Purpose

The purpose of the ConnectionManager is to enable control points to:

1. perform capability matching between source/server devices and sink/renderer device. This involves both:
    a. content-format matching (e.g., mp3 – mp3)
    b. transport (streaming) protocol matching (e.g., http – http)
2. find information about currently ongoing streams in the network, e.g.
    a. find the source device sending content to a given renderer device
    b. find the renderer devices served by a given source device or content resource
    c. find all streams going on in the network
3. setup and teardown connections between devices (when required by the streaming protocol)

2.5.2. ProtocolInfo Concept

While the UPnP Architecture describes, and prescribes, many aspects of devices that are required for a certain level of interoperability, it does not describe anything related to streaming between devices. The purpose of the ConnectionManager service is to make these aspects of devices explicit, so that control points are able to make intelligent choices, present intelligent user interfaces, and initiate (and terminate) streams between controlled devices via UPnP actions. While the actual stream of the data 'packets' occurs outside of a UPnP-defined protocol such as SOAP, SOAP is used to initiate (and terminate) the stream.

The ConnectionManager service defines the notion of "Protocol Info" as information needed by a control point in order to determine (a certain level of) compatibility between the streaming mechanisms of two UPnP controlled devices. For example, it contains the transport protocols supported by a device, for input or output, as well as other information such as the content formats (encodings) that can be sent, or received, via the transport protocols. Note that, while UPnP prescribes the use of HTTP for controlling devices via SOAP, it does not require HTTP to be used for all kinds (Audio and Video) streaming in a UPnP network.

In the context of this document, the term "protocol info" is used to describe as a string formatted as:

<protocol>':' <network>':'<contentFormat>':'<additionalInfo>

where each of the 4 elements may be a '*'. Control points can match protocol info by (protocol independent) string comparison operations on the <protocol>, <network> and <contentFormat> elements, taking into account the '*" wildcard which 'matches' with anything. The <additionalInfo> part does not need to match between source and sink. Its purpose is to convey any additional information needed to set up the out of band stream (e.g., 1394 addresses). The table below summarizes how the protocol info strings are defined for the protocols currently standardized by the ConnectionManager service, as well as for vendor-defined protocols. Section 5 provides a more detailed explanation per protocol.

Table 10: Defined Protocol Info for ConnectionManager:1

Protocol	Network	Content Format	Additional Info	Reference
`http-get`	Not needed (use '*'), since all devices supporting http are part of the same IP network.	MIME-type.	Not needed, use '*".	Section 5.1.1
`rtsp-rtp-udp`	Not needed (use '*'), since all devices supporting rtsp are part of the same IP network.	Name of RTP payload type.	Not needed, use '*".	Section 5.1.2
`internal`	IP address of the device hosting the ConnectionManager.	Vendor-defined, may be '*'.	Vendor-defined, may be '*'.	Section 5.1.3
`iec61883`	GUID of the 1394 bus' Isochronous Resource Manager.	Name standardized by IEC61883.	GUID and PCR index of the 1394 device.	Section 5.1.4
«registered ICANN domain name of vendor »	Vendor-defined, may be '*'.	Vendor-defined, may be '*'.	Vendor-defined, may be '*'.	Section 5.1.5

2.5.3. Typical Control Point Operations

This section briefly outlines some typical control point operations on a ConnectionManager service.

2.5.3.1. Establishing a new Connection

The process for establishing a streaming connection involves:

1. finding ConnectionManager services via SSDP,
2. determining compatibility between a source (sending) and a sink (receiving) device,
3. when implemented, calling the PrepareForConnection action on both source and sink devices,
4. when implemented, calling the ConnectionComplete action on both source and sink devices (after the user is done with the connection).

Because a number of these steps are better described in a larger context involving specific device types and other services as well, we refer to the 'AV Framework' document [ref to device model] for more information.

2.5.3.2. Dealing with ongoing Connections

A number of interesting scenarios require a control point to find information about all currently ongoing connections in the network, including those that it did not establish itself. This is supported by the ConnectionManager as follows. Each connection explicitly established by any control point in the network is identified by a 'connection Id' on both the source (sending) device and the sink (receiving) device. State variable 'CurrentConnectionIDs' holds a comma-separated list of these Ids. Given an Id, a control point can call GetConnectionInfo to obtain:

- The protocol info of the connection. This includes the streaming protocol and the content format.
- The 'other end' of the connection, expressed as a UDN/ServiceId pair. Using the UDN, a control point can use SSDP to find the device description of the other UPnP device involved in the connection. This way, a control point can find out, for example, that turning off a particular source device is going to affect 1 or more sink devices.
- The connection status.
- The AVTransportID of the connection, which indicates the AVTransport service instance controlling the playback and recording though the connection. This service can be used for many purposes, for example to:
    - subscribe to events in order to monitor the transport state
    - actually change the transport state, e.g., stopping or pausing an existing stream
    - obtain a URI reference to the content resource current flowing through the connection
    - obtain any meta data embedded in the content resource flowing through the connection.

  See the AVTransport service description for more details.
- The RcsID of the connection, which indicates the RenderingControl service instance controlling the rendering properties of the content. This can be used, for example, to implement a 'mute all streams' function in a control point.

2.5.4. Relation to Devices without ConnectionManagers

In some cases, it is desirable to establish a stream connection between devices where one device implements a UPnP ConnectionManager service, and the other device doesn't implement this service or isn't even a UPnP device. In such cases, a control point can only call PrepareForConnection and ConnectionComplete actions on first device. The 'PeerConnectionManager' input parameter to

PrepareForConnection is defined as the UDN of the connecting UPnP device followed by a slash ('/') and the service ID of the connecting device's ConnectionManager service. In case the connecting UPnP device has no ConnectionManager service, the service ID part of the parameter is left blank. In case the connecting device is no UPnP device (e.g., an Internet streaming server), the whole PeerConnectionManager parameter is left blank.

3. XML Service Description

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <actionList>
        <action>
            <name>GetProtocolInfo</name>
            <argumentList>
                <argument>
                    <name>Source</name>
                    <direction>out</direction>
<relatedStateVariable>SourceProtocolInfo</relatedStateVariable>
                </argument>
                <argument>
                    <name>Sink</name>
                    <direction>out</direction>
<relatedStateVariable>SinkProtocolInfo</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>PrepareForConnection</name>
            <argumentList>
                <argument>
                    <name>RemoteProtocolInfo</name>
                    <direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_ProtocolInfo</relatedStateVariable>
                </argument>
                <argument>
                    <name>PeerConnectionManager</name>
                    <direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_ConnectionManager</relatedStateVariable
>
                </argument>
                <argument>
                    <name>PeerConnectionID</name>
                    <direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_ConnectionID</relatedStateVariable>
                </argument>

                <argument>
                    <name>Direction</name>
                    <direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_Direction</relatedStateVariable>
                </argument>
                <argument>
                    <name>ConnectionID</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_ConnectionID</relatedStateVariable>
                </argument>
                <argument>
                    <name>AVTransportID</name>
```

```
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_AVTransportID</relatedStateVariable>
                </argument>
                <argument>
                    <name>RcsID</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_RcsID</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>ConnectionComplete</name>
            <argumentList>
                <argument>
                    <name>ConnectionID</name>
                    <direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_ConnectionID</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>GetCurrentConnectionIDs</name>
            <argumentList>
                <argument>
                    <name>ConnectionIDs</name>
                    <direction>out</direction>
<relatedStateVariable>CurrentConnectionIDs</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>GetCurrentConnectionInfo</name>
            <argumentList>
                <argument>
                    <name>ConnectionID</name>
                    <direction>in</direction>
<relatedStateVariable>A_ARG_TYPE_ConnectionID</relatedStateVariable>
                </argument>
                <argument>
                    <name>RcsID</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_RcsID</relatedStateVariable>
                </argument>
                <argument>
                    <name>AVTransportID</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_AVTransportID</relatedStateVariable>
                </argument>
                <argument>
                    <name>ProtocolInfo</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_ProtocolInfo</relatedStateVariable>
                </argument>
                <argument>
                    <name>PeerConnectionManager</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_ConnectionManager</relatedStateVariable
>
                </argument>
                <argument>
```

```
                    <name>PeerConnectionID</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_ConnectionID</relatedStateVariable>
                </argument>
                <argument>
                    <name>Direction</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_Direction</relatedStateVariable>
                </argument>
                <argument>
                    <name>Status</name>
                    <direction>out</direction>
<relatedStateVariable>A_ARG_TYPE_ConnectionStatus</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
    </actionList>
    <serviceStateTable>
        <stateVariable sendEvents="yes">
            <name>SourceProtocolInfo</name>
            <dataType>string</dataType>
        </stateVariable>
        <stateVariable sendEvents="yes">
            <name>SinkProtocolInfo</name>
            <dataType>string</dataType>
        </stateVariable>
        <stateVariable sendEvents="yes">
            <name>CurrentConnectionIDs</name>
            <dataType>string</dataType>
        </stateVariable>
        <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_ConnectionStatus</name>
            <dataType>string</dataType>
            <allowedValueList>
                <allowedValue>OK</allowedValue>
                <allowedValue>ContentFormatMismatch</allowedValue>
                <allowedValue>InsufficientBandwidth</allowedValue>
                <allowedValue>UnreliableChannel</allowedValue>
                <allowedValue>Unknown</allowedValue>
            </allowedValueList>
        </stateVariable>
        <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_ConnectionManager</name>
            <dataType>string</dataType>
        </stateVariable>
        <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_Direction</name>
            <dataType>string</dataType>
            <allowedValueList>
                <allowedValue>Input</allowedValue>
                <allowedValue>Output</allowedValue>
            </allowedValueList>
        </stateVariable>
        <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_ProtocolInfo</name>
            <dataType>string</dataType>
        </stateVariable>
        <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_ConnectionID</name>
            <dataType>i4</dataType>
```

```
        </stateVariable>
        <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_AVTransportID</name>
            <dataType>i4</dataType>
        </stateVariable>
        <stateVariable sendEvents="no">
            <name>A_ARG_TYPE_RcsID</name>
            <dataType>i4</dataType>
        </stateVariable>
    </serviceStateTable>
</scpd>
```

4. Test

No semantics tests have been defined for this service.

5. Appendix A – Protocol-specifics

5.1.1. Application to 'HTTP GET' - streaming

5.1.1.1. ProtocolInfo definition

Streaming data via HTTP 'GET' is defined by the Internet standard Request For Comment document entitled Hypertext Connection Protocol – HTTP/1.1 (http://www.ietf.org/rfc/rfc2616.txt). While, it is certainly possible to use other HTTP methods such as PUT or POST, this document focuses on the HTTP GET method. The protocol part of the protocol info is `http.` The 'network' part of the protocol info string is not used for the HTTP case, an asterisk ('*') is used instead. The content format for http-get is described by a MIME type, see http://www.ietf.org/rfc/rfc1341.txt.

An example of protocol information for `http-get`, in this case referring to an audio file, is:

```
http-get:*:audio/mpeg:*
```

5.1.1.2. Implementation of ConnectionManager::PrepareForConnection

Since HTTP is a stateless protocol, there is typically very little to do in the PrepareForConnection call. On the MediaRenderer device (the receiving end of the HTTP stream), the PrepareForConnection call returns an instance to the AVTransport instance to be used for transport control.

This action is optional for the HTTP protocol.

5.1.1.3. Implementation of ConnectionManager::ConnectionComplete

To manually teardown an ongoing Connection, a control point may invoke ConnectionComplete actions on either the source or sink device. For HTTP Connections, many of the underlying TCP/IP socket conventions for cleanup are utilized. In the case of a manual teardown via the method ConnectionComplete, the device simply closes the TCP/IP socket used by the AVTransport associated with the connection.

On the UPnP level, this will appear as an (evented) change in state variable CurrentConnectionIDs.

This action is optional for the HTTP protocol.

5.1.1.4. Automatic Connection Cleanup

Since control points may establish Connections, and then leave the UPnP network forever, protocols supported by the ConnectionManager need to have a built-in automatic mechanism to 'cleanup' stale connections. For HTTP Connections, automatic cleanup should be performed by the AVTransport instance.

On the UPnP level, this will appear as an (evented) change in state variable CurrentConnectionIDs.

5.1.2. Application to RTSP/RTP/UDP streaming

5.1.2.1. ProtocolInfo definition

Streaming data via RTSP is defined by the Internet standard Request For Comment document entitled Real Time Streaming Protocol. (http://www.ietf.org/rfc/rfc2326.txt). The actual Audio/Video data packets are sent out-of-band with respect to RTSP. RTSP does not require a particular protocol for this. Since usually RTP (http://www.ietf.org/rfc/rfc1889.txt) over UDP is used, we will define the protocol for RTSP-based streams as `rtsp-rtp-udp`. This ensures that two ConnectionManagers that can send and receive RTSP

also send and receive using the same Audio/Video data Connection protocol. RTP packets contain a standardized 7-bit payload type identifier, see http://www.iana.org/assignments/rtp-parameters or http://www.ietf.org/rfc/rfc1890.txt Each payload type has a unique encoding name. This payload type name is used as the "content-format" of the protocol info string.

An example of protocol information for RTSP over RTP over UDP with MPEG video payload is:

```
rtsp-rtp-udp:*:MPV:*
```

5.1.2.2. Implementation of ConnectionManager::PrepareForConnection

Since RTSP sessions are maintained by the AVTransport service, there is typically very little to do in the PrepareForConnection call. On the MediaRenderer device (the receiving end of the RTP stream), the PrepareForConnection call returns an instance to the AVTransport.

This action is optional for the RTSP/RTP/UDP protocol.

5.1.2.3. Implementation of ConnectionManager::ConnectionComplete

To manually teardown an ongoing RTSP connection, a control point may invoke ConnectionComplete actions on either the source or sink device. For RTSP sessions, many of the underlying socket conventions for cleanup are utilized. In the case of a manual teardown via the method ConnectionComplete, the device simply closes the RTSP session used by the AVTransport associated with the connection.

On the UPnP level, this will appear as an (evented) change in state variable CurrentConnectionIDs.

This action is optional for the RTSP/RTP/UDP protocol.

5.1.2.4. Automatic Connection Cleanup

Since control points may establish Connections, and then leave the UPnP network forever, protocols supported by the ConnectionManager need to have a built-in automatic mechanism to 'cleanup' stale connections. For RTSP Connections, automatic cleanup should be performed by the AVTransport instance.

On the UPnP level, this will appear as an (evented) change in state variable CurrentConnectionIDs.

5.1.3. Application to device-internal streaming

For the purpose of this service definition we define an 'internal' connection as a connection within a single device. An example of such a connection is between a Tuner subsystem and a Display subsystem in a conventional TV. Since this connection is internal to the device, no streaming data will flow on the UPnP network, and the actual content-format used inside the device can be proprietary. The resulting protocol info and content-URI that need to be defined for these types of connections can therefore be very simple.

An internal connection shall use protocol name '`internal`'. Within this protocol scope the *network identifier* is defined as the device's IP-address, as a string, in the well-known dotted decimal notation.

An example of protocol information for `internal` is:

```
internal:161.88.59.212:mpeg2:to-local-display
```

The implementation of the 'PrepareForConnection' and 'ConnectionComplete actions for this protocol type is proprietary (vendor specific).

5.1.4. Application to IEC61883 streaming

5.1.4.1. ProtocolInfo Definition

The basis for real time data transmission on the IEEE 1394 bus using the `iec61883` protocol is the Common Isochronous Packet (CIP) which consists of a CIP header and data blocks embedded in an IEEE 1394 compliant isochronous packet. The stream types include all content formats supported by the family of IEC 61883 Standards. These formats are uniquely identified by the FMT and FDF values in the CIP header, The following table lists the formats supported by the IEC 61883-2 to 5 International Standards and by IEC 61883-6 PAS (Publicly Available Specification *i.e. not yet fulfilling all requirements for a standard*).

Content Format for Protocol: "iec61883"	**Description**
UNKNOWN_STREAM	
DVCR_STD_DEF_525_60	525_60 525-line system 29.97 Hz
DVCR_STD_DEF_625_50	625_50 625-line system 25 Hz
DVCR_STD_DEF_HI_COMPRESS_525_60	
DVCR_STD_DEF_HI_COMPRESS_625_50	
DVCR_HI_DEF_1125_60	
DVCR_HI_DEF_1250_50	
SMPTE_D7_525_60	
SMPTE_D7_625_50	SMPTE V16.8-3D
MPEG2_TS	
AUDIO_MUSIC_8_24_IEC_60958	Audio and music 32-bit data consisting of 8-bit label and 24-bit data
AUDIO_MUSIC_8_24_RAW_AUDIO	
AUDIO_MUSIC_8_24_MIdI	

The network identifier for the `iec61883` protocol uniquely identifies a set of connected IEEE 1394 devices. It is defined as a bin.hex encoding of the GUID (globally unique id) of the 1394 Isochronous Resource Manager node. This uniquely identifies a single set of physically connected 1394 devices. This identification is not persistent, and will, in general, change when 1394 devices are added to or removed from the 1394 network. These changes will lead to changes in the ProtocolInfo state variable, and, through eventing, interested control points will be notified of the new streaming possibilities of the new 1394 network segmentation.

IEC61883 connections are setup between iPCRs (input Plug Control Registers) and oPCRs (output Plug Control Registers). A content item is Connectioned through an oPCR to one or more iPCRs on a different device. An IEC61883 device can have 0 or more iPCRs and oPCRs.

The *additionalInfo* field identifies the PCR in the IEC61883 network, and is defined as follows:

<GUID>';'<PCR-index>'

where

- <GUID> = bin.hex encoding of the device's `node_vendor_id` and `chip_id` (2 quadlets, together also referred to as GUID)

- <PCR-index> = zero-based integer index identifying the plug within the device

An example of protocol information for IEC61883 is:

```
iec61883:0000f00200001114:MPEG2_TS:00ba0091c9231222;0
```

5.1.4.2. Implementation of ConnectionManager::PrepareForConnection

In order to manage isochronous data transmission, IEC 61883 defines the concept of plug and specialized registers called MPR (Master Plug Register) and PCR (Plug Control Register). These registers are used to initiate and stop transmissions. The set of procedures to control the real time data flow by manipulating the PCRs is called CMP (Connection Management Procedures). Data transmission between devices is possible when an output plug on the source device is connected to an input plug on the sink device via an isochronous channel. The data flow from a source device is controlled by the oMPR (output Master Plug Register) of the device and one oPCR (output PCR). Similarly, the data flow to a sink device is controlled by the iMPR (input MPR) and one iPCR (input PCR). The address map for these registers is well defined in conformance with ISO/IEC 13213 (ANSI/IEEE 1212). Devices can modify PCR values of remote nodes using asynchronous transactions.

After a control point finds a pair of compatible ConnectionManagers, the next step is to invoke UPnP PrepareForConnection actions on the ConnectionManagers on both source and sink devices. The IEC61883 connection will be established by the *sink* device. Given the protocol info it can locate the 1394 address of the source device (its GUID is part of the 'additional info' field of the protocol info string), and program the appropriate oPCR register to initiate the streaming. The sink device is free to choose any of its own iPCRs. The sink device shall follow the exact procedure defined by IEC61883, which includes the allocation of 1394 bandwidth and a 1394 channel.. Upon subsequent 1394 bus resets, the *sink* device (the device that established the connection) shall try to restore any existing connections that it has established.

Since 1394 is a 'push' protocol, it is the responsibility of the *source* device to return an AVTransport instance id for transport control (play, pause, stop, etc.).

If the protocol info references an oPCR that is already in use, two situations occur:

- the same content-format already being streamed via the oPCR. In this case, the sink device will perform an IEC61883 *overlay* connection.
- a different content-format is already being streamed via the oPCR. In this case, the sink device will return an error.

IEC61883 broadcast-in and broadcast-out connections are not supported by the ConnectionManager.

5.1.4.3. Implementation of ConnectionManager::ConnectionComplete

To manually teardown an ongoing Connection, or to cleanup a Connection that has finished, a control point may invoke ConnectionComplete actions on both source and sink devices. It is the responsibility of the *sink* device (the device that established the connection) to perform the IEC61883 release connection procedure, by:

- Modifying corresponding fields of source oPCR and sink iPCR according to CMP procedures.
- Deallocate 1394 resources: Bandwidth and Channel if oPCR becomes unconnected (i.e. breaking last connection)

On the UPnP level, this will appear as an (evented) change in state variable CurrentConnectionIDs.

IEC61883 broadcast-in and broadcast-out connections are not supported by the ConnectionManager.

5.1.4.4. Automatic Connection Cleanup

Since control points may establish Connections, and then leave the UPnP network forever, protocols supported by the ConnectionManager need to have a built-in automatic mechanism to 'cleanup' stale

connections. For the IEC61883 protocol, an established connection will continue forever, until there is a so called *bus reset*. A bus reset will occur when there is a change in the physical network topology, for example, the network is split, joined with another network, or a device goes offline. After a bus reset, all 1394 resources are released, and all devices that established IEC61883 connections have 1 second to re-establish them. Hence, the ConnectionManager on the sink device needs to check after a bus reset whether the source device is still on the network, and if not, cleanup any internal state referring to this connection. On the UPnP level, this will appear as an (evented) change in state variable CurrentConnectionIDs.

5.1.5. Application to vendor-specific streaming

To allow vendors to use their vendor-specific streaming protocols in a UPnP network in a controlled way, the ConnectionManager defines the generic protocol info format for such protocols. The idea is to make the <protocol> part of the string unique, by requiring the use of the vendor's registered ICANN (Internet) domain name (similar to its use in vendor-specific UPnP service- and device-types). The remaining fields of the protocol info string (networkID, content-format and additional-info) are all vendor-specific, and may be wildcards ('*').

An example of vendor-specific protocol information is:

```
company.com:*:company-format-A:optional-setup-info
```

The implementation of the 'PrepareForConnection' and 'ConnectionComplete actions for this protocol type is proprietary (vendor specific).